# 基于LSTM的中文短文本分类

该项目使用PyTorch和gensim框架训练了一个基于LSTM的中文短文本分类模型。需要安装python3和PyTorch以及gensim。

数据准备部分：

需要在目录内准备训练数据train\_txt和验证数据test\_txt以及word2vec\_model\_general词向量数据

class Mydata(object):

    def \_\_init\_\_(self,train=True):

        if train:

            self.data = train\_data

            self.label = train\_label

        else:

            self.data = test\_data

            self.label = test\_label

    def \_\_getitem\_\_(self,index):

        return self.data[index],self.label[index]

    def \_\_len\_\_(self):

        return len(self.label)

def process\_data(data):

    data\_list = []

    for text in data:

        text = [word2ix[word] for word in text]

        if len(text)<=15:

            text.extend((16-len(text))\*[word2ix['pad']])

        else:

            text = text[0:15] + [word2ix['pad']]

        data\_list.append(text)

    return data\_list

#从文件train\_txt中读取训练数据

    with open("train\_txt",encoding='utf8') as f1:

        train = []

        train\_ = []

        for line in f1:

            line = line.strip()

            line = line.split(" ")

            train.append(line)

    #从文件test\_txt中读取测试数据

    with open("test\_txt",encoding='utf8') as f2:

        test = []

        test\_ = []

        for line in f2:

            line = line.strip()

            line = line.split(" ")

            test.append(line)

    train1 = []

    train1.extend(train)

    train1.extend(test)

    #建立词典

    dictionary = gensim.corpora.Dictionary(train1)

    #建立词语和id之间的关联，{词，对应的id}

    word2ix = dictionary.token2id

    # ix2word = dictionary.id2token

    ix2word = {value:key for key,value in word2ix.items()}

    #设置最后一个词

    word2ix['pad'] = len(word2ix)

    ix2word[len(word2ix)-1] = 'pad'

    #定制训练和测试的数据格式

    train\_data = process\_data(train)

    test\_data = process\_data(test)

    train\_label = [0]\*8000+[1]\*8000+[2]\*8000+[3]\*8000+[4]\*8000+[5]\*8000+[6]\*8000+[7]\*8000

    test\_label = [0]\*2000+[1]\*2000+[2]\*2000+[3]\*2000+[4]\*2000+[5]\*2000+[6]\*2000+[7]\*2000

    train\_data = torch.Tensor(train\_data)

    test\_data = torch.Tensor(test\_data)

    train\_label = torch.Tensor(train\_label)

    test\_label = torch.Tensor(test\_label)

    torch.manual\_seed(1) #为GPU设置随机种子，提高实验效果

    random.seed(1)

    #加载word2vec词向量并嵌入进embedding\_matrix

    word2vec\_model = gensim.models.Word2Vec.load('word2vec\_model\_general')

    embedding\_matrix = np.zeros((69084,128))

    for word,i in word2ix.items():

        if word != 'pad':

            embedding\_matrix[i] = word2vec\_model.wv[word]

        else:

            embedding\_matrix[i] = np.array([random.normalvariate(0,1) for j in range(128)],dtype='float32')

    embedding\_matrix = torch.Tensor(embedding\_matrix)

训练部分

class MYLSTM(nn.Module):

    def \_\_init\_\_(self):

        super(MYLSTM,self).\_\_init\_\_()

        self.embedding = nn.Embedding.from\_pretrained(embedding\_matrix,freeze=True)

        self.lstm = nn.LSTM(128,64,2,bidirectional=True)

        self.linear = nn.Linear(128,8)

    def forward(self,x):

        batch\_size,seq\_len = x.shape

        x = self.embedding(x)

        h0 = c0 = torch.zeros(4,batch\_size,64).cuda()

        x = x.transpose(1,0).contiguous()

        x,(h,c) = self.lstm(x,(h0,c0))

#         print(h)

        x = self.linear(x[15,:,:])

        return x

def cal\_acc(model,flag=True):

    if flag:

        mydata = Mydata()

    else:

        mydata = Mydata(False)

    dataloader = DataLoader(mydata,batch\_size=64,shuffle=False)

    total = 0

    for i,(data,label) in enumerate(dataloader):

        data = data.long().cuda()

        label = label.long().cuda()

        results = model(data)

        \_,top\_indexs = torch.topk(results,1,1)

        top\_indexs = top\_indexs.transpose(1,0).contiguous()

        total += (top\_indexs==label).sum().item()

    if flag:

        train\_acc = float(total)/64000

        print("train\_acc:{}".format(train\_acc))

        return train\_acc

    else:

        test\_acc = float(total)/16000

        print("test\_acc:{}".format(test\_acc))

        return test\_acc

#定义LSTM

    lstm1 = MYLSTM()

    lstm1.cuda()

    epochs=15

    batch\_size=64

    optimizer = torch.optim.Adam(lstm1.parameters(),lr=0.001,weight\_decay=0.0002)

    schduler = torch.optim.lr\_scheduler.ExponentialLR(optimizer,gamma=0.8,last\_epoch=-1)

    criterion = torch.nn.CrossEntropyLoss()

    criterion.cuda()

    mydata = Mydata()

    dataloader = DataLoader(mydata,batch\_size=batch\_size,shuffle=True)

    #weight\_decay 0.0002

    #训练lstm

    loss\_list = []

    acc\_train = []

    acc\_test = []

    for j in range(epochs):

        for i,(data,label) in enumerate(dataloader):

            data = data.long().cuda()

            label = label.long().cuda()

            optimizer.zero\_grad()

            output = lstm1(data)

            loss = criterion(output,label.view(-1))

            loss.backward()

            optimizer.step()

    #       schduler.step()

        loss\_list.append(loss)

        print("第{}epoch训练的loss：{}".format(j,loss))

        train\_acc = cal\_acc(lstm1)

        test\_acc = cal\_acc(lstm1,False)

        acc\_train.append(train\_acc)

        acc\_test.append(test\_acc)

        schedule

训练结果

第0epoch训练的loss：0.7694088220596313

train\_acc:0.8951875

test\_acc:0.8799375

第1epoch训练的loss：0.22125577926635742

train\_acc:0.91315625

test\_acc:0.887375

第2epoch训练的loss：0.2639394998550415

train\_acc:0.929515625

test\_acc:0.893625

第3epoch训练的loss：0.3165263533592224

train\_acc:0.93834375

test\_acc:0.893125

第4epoch训练的loss：0.2176312804222107

train\_acc:0.94971875

test\_acc:0.8955625

第5epoch训练的loss：0.12466021627187729

train\_acc:0.954515625

test\_acc:0.8934375

第6epoch训练的loss：0.16755922138690948

train\_acc:0.96459375

test\_acc:0.8953125

第7epoch训练的loss：0.09698238223791122

train\_acc:0.96996875

test\_acc:0.8920625

第8epoch训练的loss：0.12087912857532501

train\_acc:0.973671875

test\_acc:0.8929375

第9epoch训练的loss：0.15679828822612762

train\_acc:0.977015625

test\_acc:0.8924375

第10epoch训练的loss：0.23605412244796753

train\_acc:0.979390625

test\_acc:0.893375

第11epoch训练的loss：0.10083549469709396

train\_acc:0.981953125

test\_acc:0.8906875

第12epoch训练的loss：0.07788564264774323

train\_acc:0.983328125

test\_acc:0.88875

第13epoch训练的loss：0.08265361189842224

train\_acc:0.984296875

test\_acc:0.8896875

第14epoch训练的loss：0.030310988426208496

train\_acc:0.985203125

test\_acc:0.8885625

r.step()